МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Исследование случайного бинарного дерева поиска

Студентка гр. 8304 _____ Мельникова О.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Мельникова Ольга Александровна

Группа 8304

Тема работы: исследование случайного бинарного дерева поиска

Исходные данные:

Написать программу для создания структур данных, обработки и генерации входных данных, использовать их для измерения количественных характеристик случайного БДП, сравнить экспериментальные результаты с теоретическими.

Содержание пояснительной записки:

- Содержание
- Введение
- Описание структур данных
- Описание алгоритма
- Тестирование
- Исследование
- Исходный код
- Выводы
- Использованные источники

Дата выдачи задания: 11.10.2019

Дата сдачи реферата:

Дата защиты реферата: 17.12.2019

Студентка _____ Мельникова О.А.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В данной работе были созданы две программы на языке программирования C++, для генерации, обработки данных и вывода результатов: экспериментальных характеристик случайного БДП. Результаты были проанализированы. Для лучшего понимания кода, в нем было приведено большое количество комментариев. Также была проведена его оптимизация с целью экономии выделяемой в процессе работы памяти и улучшения быстродействия программы.

SUMMARY

In this paper, two programs were created in the C++ programming language to generate, process data and output results: experimental characteristics of random BDP. The results were analyzed. For a better understanding of the code, it was given a large number of comments. It was also optimized in order to save memory allocated in the process and improve the performance of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной курсовой работы является реализация случайного бинарного дерева поиска. Хранение данных в удобном формате с эффективной реализацией набора операций в т.ч. таких, как поиск заданного элемента, добавление (вставка) заданного элемента, удаление заданного элемента, упорядочение, является актуальным вопросом в программировании. Существует множество различных структур данных, поэтому задачей является исследовать различные числовые характеристики случайного БДП с худшем и среднем случае, а также сравнить с результатами полученными на идеально сбалансированном дереве.

1. ОПИСАНИЕ СТРУКТУР ДАННЫХ

1.1. Случайное БДП

Бинарное дерево поиска (БДП) — дерево, для которого выполняется условие:

Пусть *k* — значение ключа в узле, тогда в левом поддереве этого узла нет узлов с ключами, большими или равными *k*, а в правом поддереве — нет узлов с ключами, меньшими или равными *k*.

Случайное БДП вид с более экономным добавлением и исключением узлов, его структура полностью зависит от того («случайного») порядка, в котором элементы расположены во входной последовательности (во входном файле). В качестве примера рассмотрена последовательность из четырех элементов a, b, c, d. Имеется 4! = 24 перестановки, следовательно 24 варианта входной последовательности, они указаны на рисунке 1.

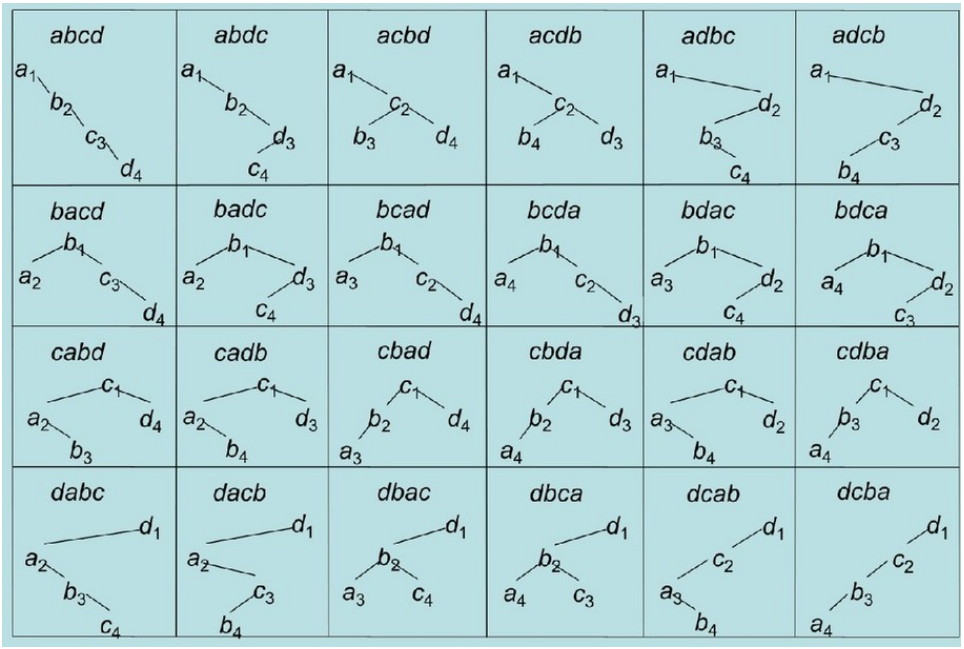

Рисунок №1 — Варианты случайного БДП из четырех элементов.

1.2. Идеально сбалансированное дерево

Идеально сбалансированным называется такое бинарное дерево T, что для каждого его узла *x* справедливо соотношение

$$|n_L(x) - n_R(x)| \le 1,$$

где $n_L(x)$ - количество узлов в левом поддереве узла *x*, а $n_R(x)$ - количество узлов в правом поддереве узла *x*.

Примеры идеально сбалансированных деревьев представлены на рисунке 2.

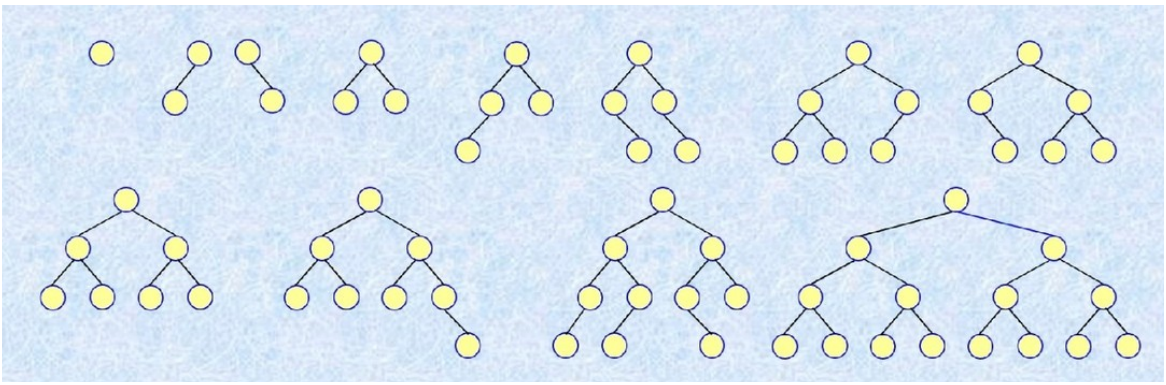

Рисунок №2 — Идеально сбалансированные деревья

2. ОПИСАНИЕ АЛГОРИТМОВ

2.1. Операции поиска, удаления, добавления элемента и алгоритм построения случайного БДП

Операция поиска заданного элемента x в непустом БДП производится рекурсивно:

- Если $k(x) = k(b)$, то элемент x находится в корне дерева b.
- Если $k(x) < k(b)$, то продолжить поиск в левом поддереве $Left(b)$.
- Если $k(x) > k(b)$, то продолжить поиск в правом поддереве $Right(b)$.
- Если выбранное дерево оказывается пустым, то элемента x в дереве нет.

Очевидно, что время поиска зависит от положения искомого узла в дереве и в худшем случае пропорционально высоте дерева.

Создание БДП было реализовано с помощью метода makeFromFile, который из входного потока считывает элементы и добавляет их в БДП методом SearchAndInsert.

Добавление элемента происходит в случае его неудачного поиска в узлах дерева. Если дерево изначально пустое, то сразу заполняются поля первого узла. Вставка зависит от значения элемента, если он меньше текущего корня, то продолжается поиск в левом поддереве, если больше, то в правом, если же значение value текущего узла равно искомому элементу, то в этом узле увеличивается count.

Ниже описано как происходит удаление элемента в случайном БДП. В зависимости от значения элемента происходит поиск либо в левом, либо в правом поддереве. При нахождении элемента происходит его удаление. Проще всего его удалить, если этот элемент находится в листе дерева. Тогда данный лист непосредственно удаляется. Если же удаляемый элемент находится во внутреннем узле b, то в ситуации когда существует правое поддерево, алгоритм находит минимальный элемент правого поддерева, рекурсивно удаляет его и заменяет им содержимое узла b. Если правого поддерева нет, то находится

максимальный элемент левого поддерева, рекурсивно удаляется и содержимое узла b заменяется им. При этом если у найденного минимума или максимума есть поддерево (левое, если максимум и правое, если минимум), то оно не удаляется, а первый его узел встает на место удаленного элемента.

2.2. Операция поиска и построение идеально сбалансированного дерева

Поиск в идеально сбалансированном дереве аналогичен поиску в БДП, поскольку построение такого дерева для удобства основано на упорядоченной последовательности данных. Идея рекурсивного алгоритма представлена на рисунке 3.

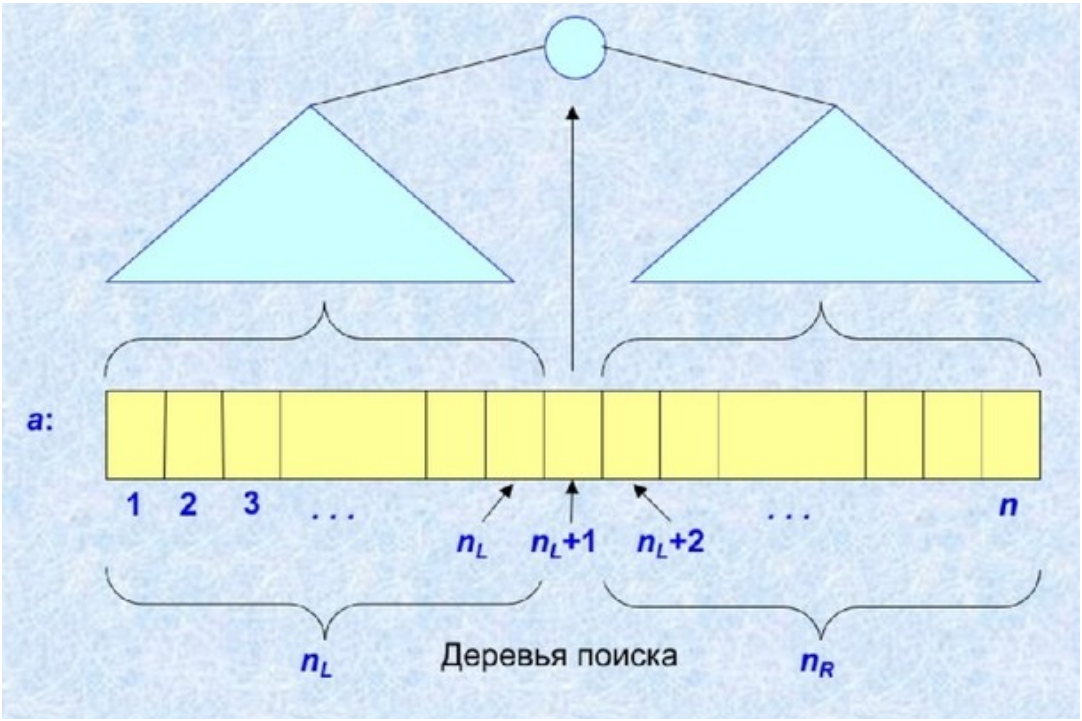

Рисунок №3 — Построение идеально сбалансированного дерева

Здесь $n_L = n \operatorname{div} 2$ и $n = n_L + n_R + 1$.

Алгоритм строит такие идеально сбалансированные деревья, что при $n_L(x) != n_R(x)$ именно левое поддерево содержит на один узел больше.

2.3. Генерация входных и получение выходных данных

Данные генерируются с увеличением числа элементов на степень двойки начиная с 1024 пятнадцать раз в отдельные файлы с помощью функции rand(). Из-за возможного повторения элементов количество узлов дерева может отличаться от входного количества, поэтому выводится количество элементов в дереве, равное числу узлов в случайном БДП в худшем случае. Кроме того, после создания случайного БДП из входных данных, идеально сбалансированного дерева из упорядоченной последовательности входных данных и случайного БДП №2 для худшего случая тоже из упорядоченной последовательности, для анализа подсчитывается среднее кол-во сравнений, среднее время поиска элемента, высота для каждого из трех деревьев по всем созданным файлам. Упорядоченная последовательность получается в результате ЛКП обхода случайного БДП. Среднее время выводится в мс, а среднее число сравнений — не целое число.

3. ОПИСАНИЕ КЛАССОВ

3.1. RandomBinaryTree

1. *T value;*

   Хранит значение.

2. *unsigned int count;*

   Хранит количество повторяющихся элементов.

3. *std::unique_ptr<BinaryTree<T>> leftLeaf;*

   Указатель на левое поддерево.

4. *std::unique_ptr<BinaryTree<T>> rightLeaf;*

   Указатель на правое поддерево.

5. *void makeFromVector(std::ifstream& fin, std::unique_ptr<BinaryTree<T>>& tree)*

   Метод предназначен для создания дерева из упорядоченой последовательности элементов.

6. *void makeFromFile(std::ifstream& fin, std::unique_ptr<BinaryTree<T>>& tree)*

   Метод предназначен для создания дерева из входной последовательности элементов.

7. *void SearchAndInsert(T& info, std::unique_ptr<BinaryTree<T>>& tree)*

   Метод предназначен для поиска и вставки элемента.

8. *void WriteToVector(std::ofstream& fout, std::unique_ptr<BinaryTree<T>>& tree)*

   Запись элементов в массив по возрастанию.

9. *int Search(T& info, long& steps, std::unique_ptr<RandomBinaryTree<T>>& tree)*

   Метод предназначен для нахождения элемента и возврата числа сравнений.

10. *long GetDepth(std::unique_ptr<RandomBinaryTree<T>>& tree)*

    Метод предназначен для нахождения высоты дерева.

3.2. IdealBinaryTree

11. *T value;*

Хранит значение.

12. *unsigned int count;*

Хранит количество повторяющихся элементов.

13. *std::unique_ptr<IdealBinaryTree<T>> leftLeaf;*

Указатель на левое поддерево.

14. *std::unique_ptr<IdealBinaryTree<T>> rightLeaf;*

Указатель на правое поддерево.

15. *void makeFromVector(std::ifstream& fin, std::unique_ptr<IdealBinaryTree<T>>& tree)*

Метод предназначен для создания дерева из упорядоченой последовательности элементов.

16. *int Search(T& info, long& steps, std::unique_ptr<IdealBinaryTree<T>>& tree*

Метод предназначен для нахождения элемента и возврата числа сравнений.

17. *long GetDepth(std::unique_ptr<IdealBinaryTree<T>>& tree)*

Метод предназначен для нахождения высоты дерева.

3. ТЕСТИРОВАНИЕ

Запуск №1:

```
Compiling rand.cpp successful
Compiling main.cpp successful
For file: Tests/file1.txt
Высота случайного дерева:                              22
Высота идеального дерева:                              11
Высота дерева в худшем случае (кол-во):                1024
Среднее кол-во сравнений в случайном дереве:           2.5
Среднее кол-во сравнений в идеальном дереве:           9
Среднее кол-во сравнений в худшем случае:              416.6
Среднее время поиска элемента в случайном дереве:      0.001 ms
Среднее время поиска элемента в идеальном  дереве:     0.001 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.062 ms
For file: Tests/file2.txt
Высота случайного дерева:                              25
Высота идеального дерева:                              12
Высота дерева в худшем случае (кол-во):                2048
Среднее кол-во сравнений в случайном дереве:           2.5
Среднее кол-во сравнений в идеальном дереве:           10
Среднее кол-во сравнений в худшем случае:              835.3
Среднее время поиска элемента в случайном дереве:      0.001 ms
Среднее время поиска элемента в идеальном  дереве:     0.001 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.126 ms
For file: Tests/file3.txt
Высота случайного дерева:                              28
Высота идеального дерева:                              13
Высота дерева в худшем случае (кол-во):                4096
Среднее кол-во сравнений в случайном дереве:           18.2
Среднее кол-во сравнений в идеальном дереве:           11
Среднее кол-во сравнений в худшем случае:              2837.1
Среднее время поиска элемента в случайном дереве:      0.004 ms
Среднее время поиска элемента в идеальном  дереве:     0.001 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.429 ms
For file: Tests/file4.txt
Высота случайного дерева:                              36
Высота идеального дерева:                              14
Высота дерева в худшем случае (кол-во):                8192
```

```
Среднее кол-во сравнений в случайном дереве:           19.1
Среднее кол-во сравнений в идеальном дереве:           12
Среднее кол-во сравнений в худшем случае:              2602.4
Среднее время поиска элемента в случайном дереве:      0.004 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.395 ms
For file: Tests/file5.txt
Высота случайного дерева:                              32
Высота идеального дерева:                              15
Высота дерева в худшем случае (кол-во):                16384
Среднее кол-во сравнений в случайном дереве:           21
Среднее кол-во сравнений в идеальном дереве:           13
Среднее кол-во сравнений в худшем случае:              9954.4
Среднее время поиска элемента в случайном дереве:      0.005 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
Среднее время поиска элемента в худшем случае (сл.д.): 1.576 ms
For file: Tests/file6.txt
Высота случайного дерева:                              37
Высота идеального дерева:                              16
Высота дерева в худшем случае (кол-во):                32768
Среднее кол-во сравнений в случайном дереве:           18.9
Среднее кол-во сравнений в идеальном дереве:           14
Среднее кол-во сравнений в худшем случае:              18621.9
Среднее время поиска элемента в случайном дереве:      0.005 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
Среднее время поиска элемента в худшем случае (сл.д.): 3.072 ms
For file: Tests/file7.txt
Высота случайного дерева:                              39
Высота идеального дерева:                              17
Высота дерева в худшем случае (кол-во):                65532
Среднее кол-во сравнений в случайном дереве:           19
Среднее кол-во сравнений в идеальном дереве:           15
Среднее кол-во сравнений в худшем случае:              29706.8
Среднее время поиска элемента в случайном дереве:      0.006 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
Среднее время поиска элемента в худшем случае (сл.д.): 4.964 ms
For file: Tests/file8.txt
Высота случайного дерева:                              39
Высота идеального дерева:                              18
Количество элементов:                                  131067
```

```
Среднее кол-во сравнений в случайном дереве:          21.9
Среднее кол-во сравнений в идеальном дереве:          16
Среднее время поиска элемента в случайном дереве:     0.005 ms
Среднее время поиска элемента в идеальном  дереве:    0.002 ms
For file: Tests/file9.txt
Высота случайного дерева:                             50
Высота идеального дерева:                             19
Количество элементов:                                 262126
Среднее кол-во сравнений в случайном дереве:          22.5
Среднее кол-во сравнений в идеальном дереве:          17
Среднее время поиска элемента в случайном дереве:     0.005 ms
Среднее время поиска элемента в идеальном  дереве:    0.003 ms
For file: Tests/file10.txt
Высота случайного дерева:                             43
Высота идеального дерева:                             20
Количество элементов:                                 524226
Среднее кол-во сравнений в случайном дереве:          23.6
Среднее кол-во сравнений в идеальном дереве:          18
Среднее время поиска элемента в случайном дереве:     0.006 ms
Среднее время поиска элемента в идеальном  дереве:    0.003 ms
For file: Tests/file11.txt
Высота случайного дерева:                             49
Высота идеального дерева:                             21
Количество элементов:                                 1048335
Среднее кол-во сравнений в случайном дереве:          25.8
Среднее кол-во сравнений в идеальном дереве:          19
Среднее время поиска элемента в случайном дереве:     0.007 ms
Среднее время поиска элемента в идеальном  дереве:    0.003 ms
For file: Tests/file12.txt
Высота случайного дерева:                             53
Высота идеального дерева:                             22
Количество элементов:                                 2096166
Среднее кол-во сравнений в случайном дереве:          31
Среднее кол-во сравнений в идеальном дереве:          20
Среднее время поиска элемента в случайном дереве:     0.009 ms
Среднее время поиска элемента в идеальном  дереве:    0.003 ms
For file: Tests/file13.txt
Высота случайного дерева:                             57
Высота идеального дерева:                             23
Количество элементов:                                 4190316
```

```
Среднее кол-во сравнений в случайном дереве:           30.9
Среднее кол-во сравнений в идеальном дереве:           21
Среднее время поиска элемента в случайном дереве:      0.009 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
For file: Tests/file14.txt
Высота случайного дерева:                              58
Высота идеального дерева:                              24
Количество элементов:                                  8372502
Среднее кол-во сравнений в случайном дереве:           31
Среднее кол-во сравнений в идеальном дереве:           22
Среднее время поиска элемента в случайном дереве:      0.009 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
```

Запуск №2

```
Compiling rand.cpp successful
Compiling main.cpp successful
For file: Tests/file1.txt
Высота случайного дерева:                              20
Высота идеального дерева:                              11
Высота дерева в худшем случае (кол-во):                1024
Среднее кол-во сравнений в случайном дереве:           2.2
Среднее кол-во сравнений в идеальном дереве:           9
Среднее кол-во сравнений в худшем случае:              560.6
Среднее время поиска элемента в случайном дереве:      0.001 ms
Среднее время поиска элемента в идеальном  дереве:     0.001 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.092 ms
For file: Tests/file2.txt
Высота случайного дерева:                              24
Высота идеального дерева:                              12
Высота дерева в худшем случае (кол-во):                2048
Среднее кол-во сравнений в случайном дереве:           14.4
Среднее кол-во сравнений в идеальном дереве:           10
Среднее кол-во сравнений в худшем случае:              1118.5
Среднее время поиска элемента в случайном дереве:      0.003 ms
Среднее время поиска элемента в идеальном  дереве:     0.001 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.183 ms
For file: Tests/file3.txt
Высота случайного дерева:                              24
Высота идеального дерева:                              13
```

```
Высота дерева в худшем случае (кол-во):                4096
Среднее кол-во сравнений в случайном дереве:           16.7
Среднее кол-во сравнений в идеальном дереве:           11
Среднее кол-во сравнений в худшем случае:              2399.4
Среднее время поиска элемента в случайном дереве:      0.004 ms
Среднее время поиска элемента в идеальном  дереве:     0.001 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.364 ms
For file: Tests/file4.txt
Высота случайного дерева:                              34
Высота идеального дерева:                              14
Высота дерева в худшем случае (кол-во):                8192
Среднее кол-во сравнений в случайном дереве:           17.6
Среднее кол-во сравнений в идеальном дереве:           12
Среднее кол-во сравнений в худшем случае:              3551.2
Среднее время поиска элемента в случайном дереве:      0.004 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
Среднее время поиска элемента в худшем случае (сл.д.): 0.55 ms
For file: Tests/file5.txt
Высота случайного дерева:                              36
Высота идеального дерева:                              15
Высота дерева в худшем случае (кол-во):                16384
Среднее кол-во сравнений в случайном дереве:           21.9
Среднее кол-во сравнений в идеальном дереве:           13
Среднее кол-во сравнений в худшем случае:              9850.5
Среднее время поиска элемента в случайном дереве:      0.005 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
Среднее время поиска элемента в худшем случае (сл.д.): 1.554 ms
For file: Tests/file6.txt
Высота случайного дерева:                              39
Высота идеального дерева:                              16
Количество элементов:                                  32768
Среднее кол-во сравнений в случайном дереве:           19.2
Среднее кол-во сравнений в идеальном дереве:           14
Среднее время поиска элемента в случайном дереве:      0.004 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
For file: Tests/file7.txt
Высота случайного дерева:                              38
Высота идеального дерева:                              17
Количество элементов:                                  65536
Среднее кол-во сравнений в случайном дереве:           3
```

```
Среднее кол-во сравнений в идеальном дереве:          15
Среднее время поиска элемента в случайном дереве:     0.001 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
For file: Tests/file8.txt
Высота случайного дерева:                              42
Высота идеального дерева:                              18
Количество элементов:                                  131067
Среднее кол-во сравнений в случайном дереве:          22.6
Среднее кол-во сравнений в идеальном дереве:          16
Среднее время поиска элемента в случайном дереве:     0.006 ms
Среднее время поиска элемента в идеальном  дереве:     0.002 ms
For file: Tests/file9.txt
Высота случайного дерева:                              42
Высота идеального дерева:                              19
Количество элементов:                                  262124
Среднее кол-во сравнений в случайном дереве:          24.9
Среднее кол-во сравнений в идеальном дереве:          17
Среднее время поиска элемента в случайном дереве:     0.006 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
For file: Tests/file10.txt
Высота случайного дерева:                              48
Высота идеального дерева:                              20
Количество элементов:                                  524223
Среднее кол-во сравнений в случайном дереве:          26.4
Среднее кол-во сравнений в идеальном дереве:          18
Среднее время поиска элемента в случайном дереве:     0.007 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
For file: Tests/file11.txt
Высота случайного дерева:                              52
Высота идеального дерева:                              21
Количество элементов:                                  1048329
Среднее кол-во сравнений в случайном дереве:          29.3
Среднее кол-во сравнений в идеальном дереве:          19
Среднее время поиска элемента в случайном дереве:     0.008 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
For file: Tests/file12.txt
Высота случайного дерева:                              50
Высота идеального дерева:                              22
Количество элементов:                                  2096095
Среднее кол-во сравнений в случайном дереве:          26.8
```

```
Среднее кол-во сравнений в идеальном дереве:           20
Среднее время поиска элемента в случайном дереве:      0.008 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
For file: Tests/file13.txt
Высота случайного дерева:                              54
Высота идеального дерева:                              23
Количество элементов:                                  4190234
Среднее кол-во сравнений в случайном дереве:           28.1
Среднее кол-во сравнений в идеальном дереве:           21
Среднее время поиска элемента в случайном дереве:      0.008 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
For file: Tests/file14.txt
Высота случайного дерева:                              57
Высота идеального дерева:                              24
Количество элементов:                                  8372104
Среднее кол-во сравнений в случайном дереве:           29.6
Среднее кол-во сравнений в идеальном дереве:           22
Среднее время поиска элемента в случайном дереве:      0.008 ms
Среднее время поиска элемента в идеальном  дереве:     0.003 ms
```

Результаты запуска №3 представлены в таблице №1 и №2:

Таблица №1 — Первая часть результатов запуска программы

Количество узлов	Высота случайного БДП	Высота идеально сбаланс. дерева	Среднее число сравн. в случайном БДП	Среднее число сравн. в идеально сбаланс.	Среднее число сравн. в худшем случае
1024	20	11	11,8	9	580,5
2048	24	12	14	10	1155,9
4096	27	13	15,2	11	2222,6
8192	29	14	15,8	12	3327,5
16384	33	15	19,9	13	10321,1
32768	34	16	19,9	14	14477,1
65536	36	17	22,6	15	33230,9
131069	39	18	19,5	16	-
262127	48	19	21,4	17	-

524233	47	20	24,5	18	-
1048340	51	21	29,4	19	-
2096107	49	22	28,9	20	-
4190250	55	23	28,7	21	-
8372242	59	24	32	22	-
16711450	64	25	31,7	23	-

Таблица №2 — Вторая часть результатов запуска программы

Количество узлов	Среднее время поиска в случайном БДП	Среднее время поиска в идеально сбаланс. дереве	Среднее время поиска в худшем случае
1024	0,001	0,001	0,085
2048	0,003	0,001	0,175
4096	0,003	0,002	0,334
8192	0,004	0,002	0,514
16384	0,005	0,002	1,658
32768	0,005	0,002	2,368
65536	0,005	0,002	5,423
131069	0,005	0,002	-
262127	0,005	0,003	-
524233	0,006	0,003	-
1048340	0,008	0,003	-
2096107	0,009	0,003	-
4190250	0,008	0,003	-
8372242	0,009	0,003	-
16711450	0,009	0,004	-

ИССЛЕДОВАНИЕ

Проанализируем данные, представленные в таблицах выше, для этого заполним следующую таблицу (табл. №3):

Таблица №3 — Соответствие между n и $\log_2 n$

Количество узлов (n)	$\log_2 n$
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
131069	16,99996698
262127	17,99990644
524233	18,99984865
1048340	19,99967526
2096107	20,99928093
4190250	21,99860489
8372242	22,99718258
16711450	23,99433358

В теории среднее число сравнений при поиске элемента растет как $\log_2 n$ для идеально сбалансированного и $1.386 \log_2 n$ для случайного БДП, а в худшем случае n, поэтому построим график со следующими осями: ось X — $\log_2 n$, а ось Y — n. В данных осях графики для идеального случая с идеал. сбаланс. и для среднего со случайным БДП — должны быть линейны. С помощью «LibreOffice Calc» были построены дискретные графики, а также их аппроксимация. Результаты представлены на рис. №4 и №5.

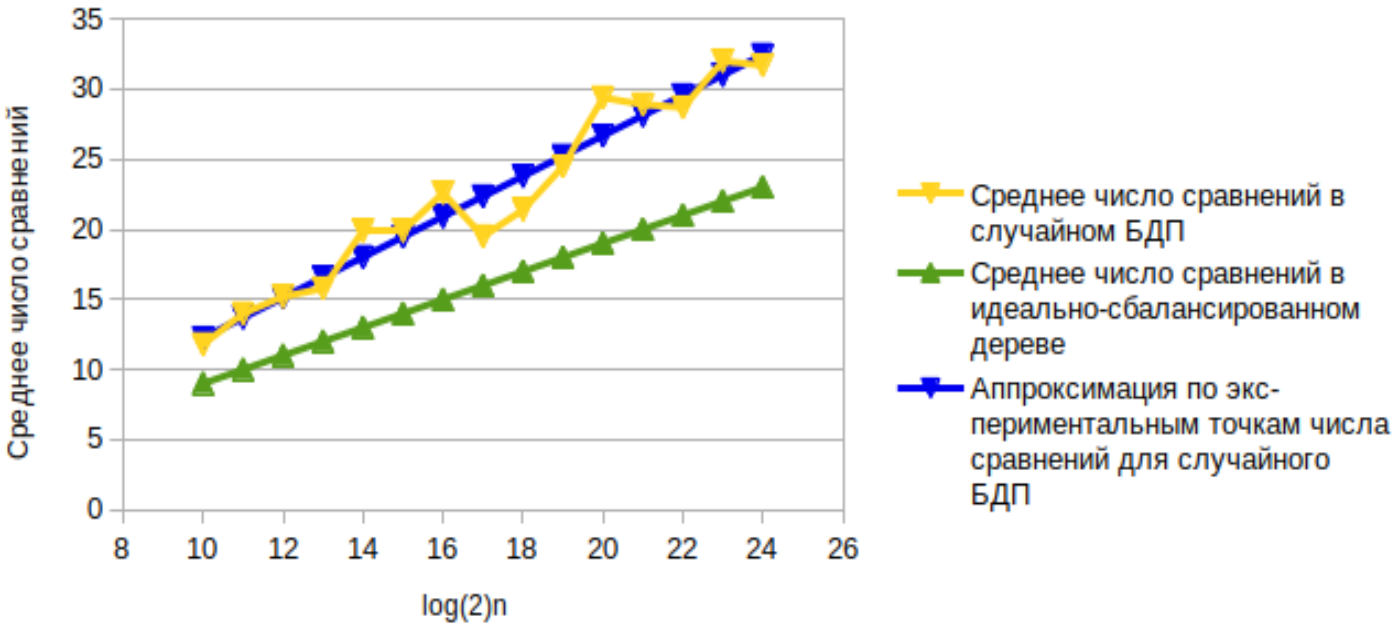

Рисунок №4 — Зависимость среднего числа сравнений от $\log_2 n$ в среднем и лучшем случае.

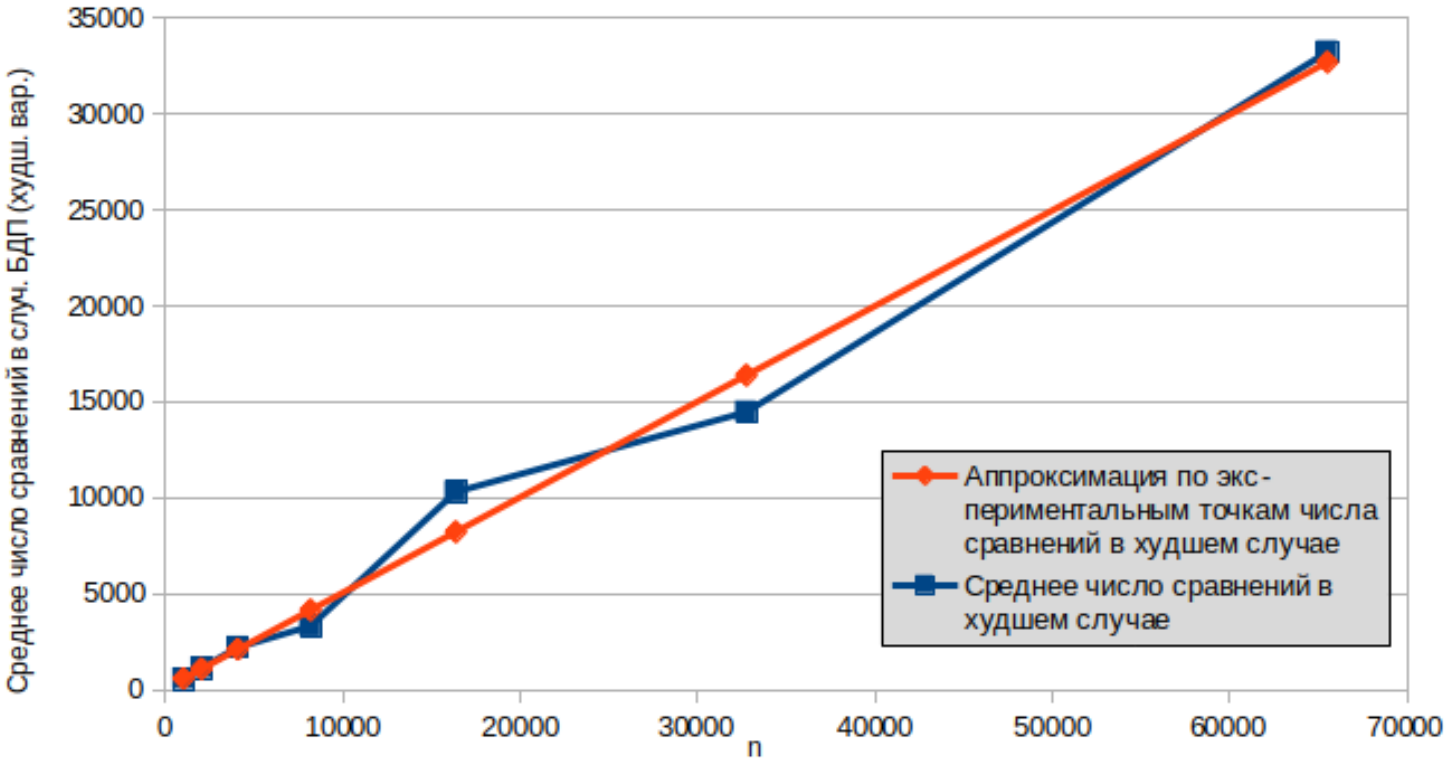

Рисунок №5 — Зависимость среднего числа сравнений от n в худшем случае.

Из рисунков выше видно, что в худшем случае график становится линейным, среднее число сравнений и время поиска (т. к. одно сравнение происходит за определенное время и время поиска пропорционально числу сравнений) линейно зависят от числа узлов в дереве, а в лучшем и среднем случае число сравнений при увеличении числа узлов растет как логарифмическая функция по основанию два, однако, как видно на рис. №4 один график выше другого. Найдем на сколько в среднем среднее число сравнений при поиске в случайном БДП больше, чем в идеально сбалансированном. Для этого поделим среднее число сравнений на значения в аппрокс. функции для каждого количества улов, а затем найдем среднее. Результаты представлены в табл. №4.

Таблица №4 — Нахождение среднего углового коэффициента для графика зависимости среднего числа сравнений от $\log_2 n$ в среднем случае.

Среднее число сравн. в случайном БДП (a)	Аппроксимация по экспериментальным точкам числа сравнений для случайного БДП (b)	$\frac{a}{b}$
11,8	12,26	1,3622
14	13,70	1,3702
15,2	15,14	1,3767
15,8	16,59	1,3822
19,9	18,03	1,3868
19,9	19,47	1,3907
22,6	20,91	1,3942
19,5	22,35	1,3971
21,4	23,80	1,3998
24,5	25,29	1,4021
29,4	26,68	1,4022
28,9	28,12	1,4061
28,7	29,56	1,4078
32	31,00	1,4092

31,7	32,44	1,4105
	Среднее $\frac{a}{b}$:	**1,3933**

По результатам расчетов, видно, что при поиске в случайном БДП среднее число сравнений всего лишь на 39% больше, чем в идеально сбалансированном дереве, что соответствует теоретическим данным. Это отражает тот факт, что среди случайных деревьев чаще встречаются хорошо сбалансированные (относительно «ровные») деревья, чем вырожденные.

Постороим график зависимости высоты дерева от числа узлов (рис. №6)

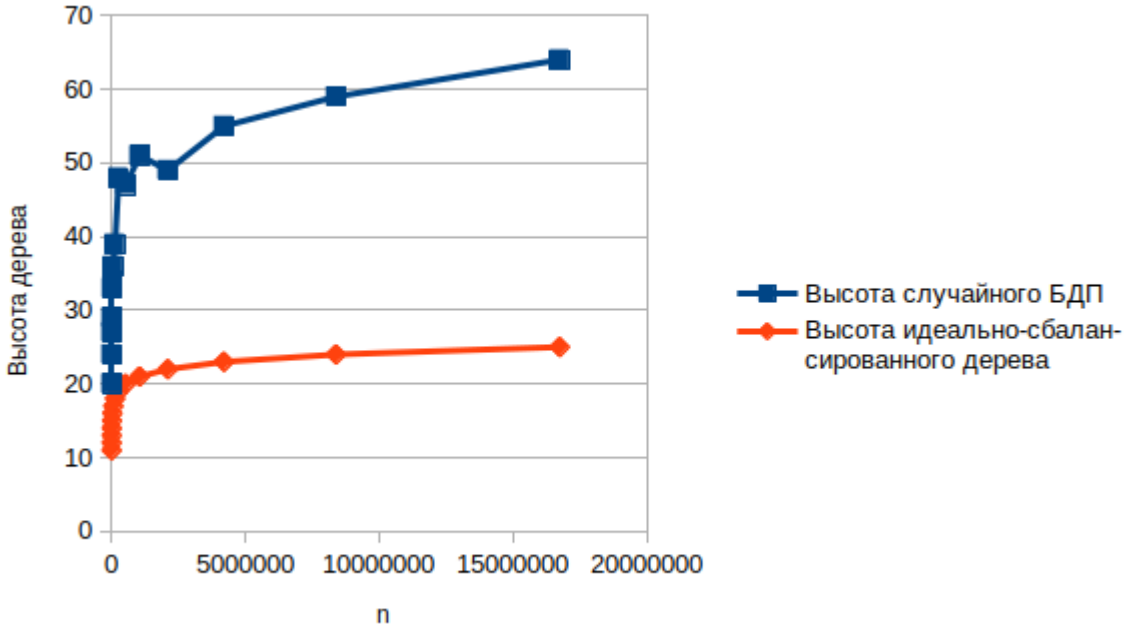

Рисунок №6 — Зависимость дерева от числа узлов в среднем и лучшем случае.

По графикам видно, случайное дерево неплохо сбалансировано и высота случайного БДП порядка $^{2\log_2 n}$ против $^{\log_2 n}$ для идеально сбалансированного дерева.

ЗАКЛЮЧЕНИЕ

В ходе работы была написана программа для генерации данных, создания таких структур данных как случайное бинарное дерево поиска, идеально сбалансированное дерево. С помощью программы были найдены экспериментальные числовые характеристики структур данных, при исследовании которых были подтверждены теоретические данные. Был получен опыт работы с дополнительными возможностями C++ и эффективной алгоритмизацией на нем. Также были закреплены знания, полученные за семестр. Исходный код находится в приложении A, B и C.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роберт Седжвик, Алгоритмы на C++, М.: Вильямс, 2011 г.

2. Martinez, Conrado; Roura, Salvador (1998), Randomized binary search trees, Journal of the ACM (ACM Press) 45 (2): 288–323

3. Reed, Bruce (2003), The height of a random binary search tree, Journal of the ACM 50 (3): 306–332

ПРИЛОЖЕНИЕ А
MAIN.CPP И ЗАГОЛОВОЧНЫЕ ФАЙЛЫ

main.cpp

```
#include "idealTree.h"
#include "randomTree.h"
#include <ctime>


int main(int argc, char* argv[]) {


    if(argc == 1)
    {
           std::cout<<"Не введен аргумент командной строки – название
файла\n";
        exit(1);
    }
    else
    {
                std::unique_ptr<IdealBinaryTree<double>>  idealTree(new
IdealBinaryTree<double>());
                std::unique_ptr<RandomBinaryTree<double>>  randTree(new
RandomBinaryTree<double>());
                 std::unique_ptr<RandomBinaryTree<double>>  badTree(new
RandomBinaryTree<double>());
          std::ifstream inputFile(argv[1]);

          std::cout << "For file: " << argv[1] << std::endl;
          if (!inputFile.is_open())
          {
              std::cout << "ERROR: file isn't open" << std::endl;
              return 0;
          }
          if (inputFile.eof())
          {
              std::cout << "ERROR: file is empty" << std::endl;
              return 0;
          }

          long searchCount = atoi(argv[2]);


          randTree->makeFromFile(inputFile, randTree);
          std::vector<double> vect;
```

```
            randTree->WriteToVector(vect, randTree);
            idealTree->makeFromVector(vect, idealTree);
            if(atoi(argv[3])) randTree->makeFromVector(vect, badTree);
            inputFile.close();


            double elem; //генерируем много элем
            long steps1 = 0;
            long steps2 = 0;
            long steps3 = 0;
            long i = 0;
            srand(time(0));

            unsigned long start_time = clock();

            while (i++ < searchCount){
                elem = rand();
                randTree->Search(elem, steps1, randTree);
            }
            unsigned long second_time  = clock();
            unsigned long time1  = (second_time - start_time)/searchCount;
            i = 0;
            while (i++ < searchCount){
                elem = rand();
                idealTree->Search(elem, steps2, idealTree);
            }
            unsigned long third_time  = clock();
            unsigned long time2  = (third_time - second_time)/searchCount;
            unsigned long time3;
            if(atoi(argv[3])){
                i = 0;
                while (i++ < searchCount){
                    elem = rand();
                    randTree->Search(elem, steps3, badTree);
                }
                unsigned long end_time  = clock();
                time3  = (end_time - third_time)/searchCount;
            }

                            std::cout   <<   "Высота   случайного   дерева:
" << randTree->GetDepth(randTree) << std::endl;
                            std::cout   <<   "Высота   идеального   дерева:
" << idealTree->GetDepth(idealTree) << std::endl;
             if(atoi(argv[3])) std::cout << "Высота дерева в худшем случае
(кол-во):              " << randTree->GetDepth(badTree) << std::endl;
```

```
                    if(!atoi(argv[3])) std::cout << "Количество элементов:
" << vect.size() << std::endl;
                std::cout << "Среднее кол-во сравнений в случайном дереве:
" << (float)steps1/searchCount << std::endl;
                std::cout << "Среднее кол-во сравнений в идеальном дереве:
" << (float)steps2/searchCount << std::endl;
                if(atoi(argv[3])) std::cout << "Среднее кол-во сравнений в
худшем случае:              " << (float)steps3/searchCount << std::endl;
                 std::cout << "Среднее время поиска элемента в случайном
дереве:          " << (float)time1 / CLOCKS_PER_SEC * 1000 << " ms" <<
std::endl;
                 std::cout << "Среднее время поиска элемента в идеальном
дереве:     " << (float)time2 / CLOCKS_PER_SEC * 1000 << " ms" << std::endl;
               if(atoi(argv[3])) std::cout << "Среднее время поиска элемента
в худшем случае (сл.д.): " << (float)time3 / CLOCKS_PER_SEC * 1000 << " ms"
<< std::endl;

            return 0;
        }
    }
```

randomTree.h

```
#pragma once
#include <iostream>
#include <string>
#include <memory>
#include <cstdlib>
#include <cstdio>
#include <fstream>
#include <vector>

// если элем удалить
template <typename T>
class RandomBinaryTree
{
public:
    T value;
    unsigned int count = 0;
    std::unique_ptr<RandomBinaryTree<T>> leftTree;
    std::unique_ptr<RandomBinaryTree<T>> rightTree;

                        void     makeFromVector(std::vector<T>&     vect,
std::unique_ptr<RandomBinaryTree<T>>& tree){
            T info;
            int i = 0;
```

```
            while(i<vect.size()){
                info = vect[i];
                i++;
                SearchAndInsert(info, tree);
            }
        }

                            void      makeFromFile(std::ifstream&      fin,
std::unique_ptr<RandomBinaryTree<T>>& tree){
            T info;
            while(fin >> info){
                SearchAndInsert(info, tree);
            }
        }
                                void       SearchAndInsert(T&       info,
std::unique_ptr<RandomBinaryTree<T>>& tree){
            if((tree->count) == 0 ) {
                tree->value = info;
                tree->count = 1;
            }else if ( info < tree->value ) {
                                if   (!tree->leftTree)   tree->leftTree   =
std::make_unique<RandomBinaryTree<T>>();
                SearchAndInsert(info, tree->leftTree);
            }else if(info>(tree->value)){
                               if   (!tree->rightTree)   tree->rightTree   =
std::make_unique<RandomBinaryTree<T>>();
                SearchAndInsert(info, tree->rightTree);
            }else (tree->count)++;
        }

                            void      WriteToFile(std::ofstream&      fout,
std::unique_ptr<RandomBinaryTree<T>>& tree) {
            if (tree->leftTree) WriteToFile(fout, tree->leftTree);
            fout << tree->value << std::endl;
            if (tree->rightTree) WriteToFile(fout, tree->rightTree);
        }
                        void     WriteToVector(std::vector<T>     &     vect,
std::unique_ptr<RandomBinaryTree<T>>& tree) {
            if (tree->leftTree) WriteToVector(vect, tree->leftTree);
            vect.push_back(tree->value);
            if (tree->rightTree) WriteToVector(vect, tree->rightTree);
        }

                           int      Search(T&      info,      long&      steps,
std::unique_ptr<RandomBinaryTree<T>>& tree){
            if (tree->value != info) {
```

```
                steps++;
                if(info < tree->value) {
                    if(tree->leftTree != NULL) {
                        return Search(info, steps, tree->leftTree);
                    }
                    else return -1;
                } else {
                    if(tree->rightTree != NULL) {
                        return Search(info, steps, tree->rightTree);
                    }
                    else return -1;
                }
            }else{
                return steps;
            }
        }

        long GetDepth(std::unique_ptr<RandomBinaryTree<T>>& tree)
        {
            long l, r;
             if (tree->leftTree)  l = GetDepth(tree->leftTree);  else l =
0;
             if (tree->rightTree) r = GetDepth(tree->rightTree); else r =
0;
            if (l > r) return l+1; else return r+1;
         }
    };
```

idealTree.h

```
#pragma once
#include <iostream>
#include <string>
#include <memory>
#include <cstdlib>
#include <cstdio>
#include <fstream>
#include <vector>
// если элем удалить
template <typename T>
class IdealBinaryTree
{
public:
    T value;
    std::unique_ptr<IdealBinaryTree<T>> leftTree;
    std::unique_ptr<IdealBinaryTree<T>> rightTree;
```

```
                            void      makeFromFile(std::ifstream&      fin,
std::unique_ptr<IdealBinaryTree<T>>& tree){
            T info;
            while(fin >> info){
                SearchAndInsert(info, tree);
            }
        }
                            void     makeFromVector(std::vector<T>&     vect,
std::unique_ptr<IdealBinaryTree<T>>& tree){
            makeFromVector(vect, 0, vect.size()-1, tree);
        }
         void makeFromVector(std::vector<T> & vect, unsigned int startPos,
unsigned int endPos, std::unique_ptr<IdealBinaryTree<T>>& tree){
            unsigned int endPosLeft, startPosRight;
            endPosLeft = (endPos+startPos) / 2;
            startPosRight = endPosLeft + 2;

            if (startPos != endPos) {
                value = vect[endPosLeft + 1];

                                    if   (!tree->leftTree)   tree->leftTree   =
std::make_unique<IdealBinaryTree<T>>();
                            makeFromVector(vect,  startPos,  endPosLeft,  tree-
>leftTree);

                if (startPosRight <= endPos) {
                                    if  (!tree->rightTree)  tree->rightTree  =
std::make_unique<IdealBinaryTree<T>>();
                            makeFromVector(vect, startPosRight, endPos, tree-
>rightTree);
                }
            }
            else {
                value = vect[endPosLeft];
            }
        }

                            void      WriteToFile(std::ofstream&      fout,
std::unique_ptr<IdealBinaryTree<T>>& tree) {
            if (tree->leftTree) WriteToFile(fout, tree->leftTree);
            fout << tree->value << std::endl;
            if (tree->rightTree) WriteToFile(fout, tree->rightTree);
        }
```

```
                        int     Search(T&     info,     long&     steps,
std::unique_ptr<IdealBinaryTree<T>>& tree){
            if (tree->value != info) {
                steps++;
                if(info < tree->value) {
                    if(tree->leftTree != NULL) {
                        return Search(info, steps, tree->leftTree);
                    }
                    else return -1;
                } else {
                    if(tree->rightTree != NULL) {
                        return Search(info, steps, tree->rightTree);
                    }
                    else return -1;
                }
            }else{
                return steps;
            }
        }

        long GetDepth(std::unique_ptr<IdealBinaryTree<T>>& tree)
        {
            long l, r;
             if (tree->leftTree)  l = GetDepth(tree->leftTree);  else l =
0;
             if (tree->rightTree) r = GetDepth(tree->rightTree); else r =
0;
            if (l > r) return l+1; else return r+1;
         }

    };
```

ПРИЛОЖЕНИЕ В

RAND.CPP

```
#include <iostream>
#include <string>
#include <memory>
#include <cstdlib>
#include <cstdio>
#include <fstream>
#include <ctime>
int main(int argc, char* argv[]){
 std::ofstream fout(argv[1]);
 long count = atoi(argv[2]);
 srand(time(0));
 int i = 0;
    while (i++ < count){
    fout << rand() << std::endl;
    }

}
```

ПРИЛОЖЕНИЕ С

TESTS.PY

```
import os
import subprocess
import shutil
out=subprocess.call(["g++ ./Source/rand.cpp -o rand"], shell=True)
outt=subprocess.call(["g++ ./Source/main.cpp -o main"], shell=True)
count = 1024;
if ((out==0) and (outt==0)):
 print("Compiling rand.cpp successful");
 print("Compiling main.cpp successful");
 i=1;
 while i<5:
    subprocess.call("./rand  Tests/file"  +  str(i)  +  ".txt  "  +
str(count), shell=True, cwd=".")
    subprocess.call("./main Tests/file" + str(i) + ".txt " + str(10) +
" " + str(1), shell=True, cwd=".")
    count*=2;
    i= i+1
 while i<15:
    subprocess.call("./rand  Tests/file"  +  str(i)  +  ".txt  "  +
str(count), shell=True, cwd=".")
    subprocess.call("./main Tests/file" + str(i) + ".txt " + str(10) +
" " + str(0), shell=True, cwd=".")
    count*=2;
    i= i+1
```